The Distributed Interactive Computing Environment

J. J. Hare, J. A. Clarke, C. E. Schmitt
U.S. Army Research Laboratory
Aberdeen Proving Ground, MD 21005

Abstract

The Distributed Interactive Computing Environment (DICE) is a toolkit that uses a powerful scripting language to combine data organization techniques, visualization utilities, and graphical user interfaces with new or existing computational codes. This combination of tools creates a full-featured High Performance Computing (HPC) application. These applications facilitate the process of HPC by taking advantage of distributed hardware and software resources, and by allowing the user to visually inspect runtime data while computations progress. This environment permits the scientist to focus on his research and not computing idiosyncrasies.

1. Introduction

Many existing scientific High Performance Computing (HPC) codes, both scalar and parallel, are designed to run in isolation, producing data that remains inaccessible until the process completes. The user edits an input file, submits the job to a batch queue, then, post-processes the results. Errors in the input may cause precious computational resources to be consumed only to produce useless results. The Distributed Interactive Computing Environment (DICE) is a toolkit that helps build full-featured, high performance computing applications from new or existing codes. DICE is designed to provide scaleable, interactive, runtime visualization from the user's desktop for codes running on a high performance platform. Because the visualization is decoupled from the running code, users can visually interrogate runtime data while the code continues to advance.

DICE consists of three major sections: data organization, visualization, and graphical user interface tools. However, a DICE application pulls a fourth major section into the scene: the computational code. A structural overview of the system is provided in figure 1. This paper generally describes how the four components of a DICE application interact and facilitate the HPC process for both the developer and the end-user.

2. Integrating a Scientific High Performance Computing Code into the Environment

Integrating an HPC code into an environment such as this could involve drastic changes to the code and require a tight coupling of the code with other components of the environment, such as visualization. This approach describes a "code-centered" application. However, DICE applications are based upon a different "data-centered" approach. This approach decouples the HPC code from the visualization, allowing both the computation and the data analysis to simultaneously act upon the data without impeding on either component's progress. While this method requires that a copy of the data be stored in a data buffer, it does not significantly inhibit the performance of the computational code.

A data buffer and other data organization utilities are maintained as the core of the entire architecture and will be described in more detail later. A computational code interacts with this data buffer via the DICE Application Interface (DAI). This interface services requests from the user for data updates, and at specified intervals writes a copy of the data to

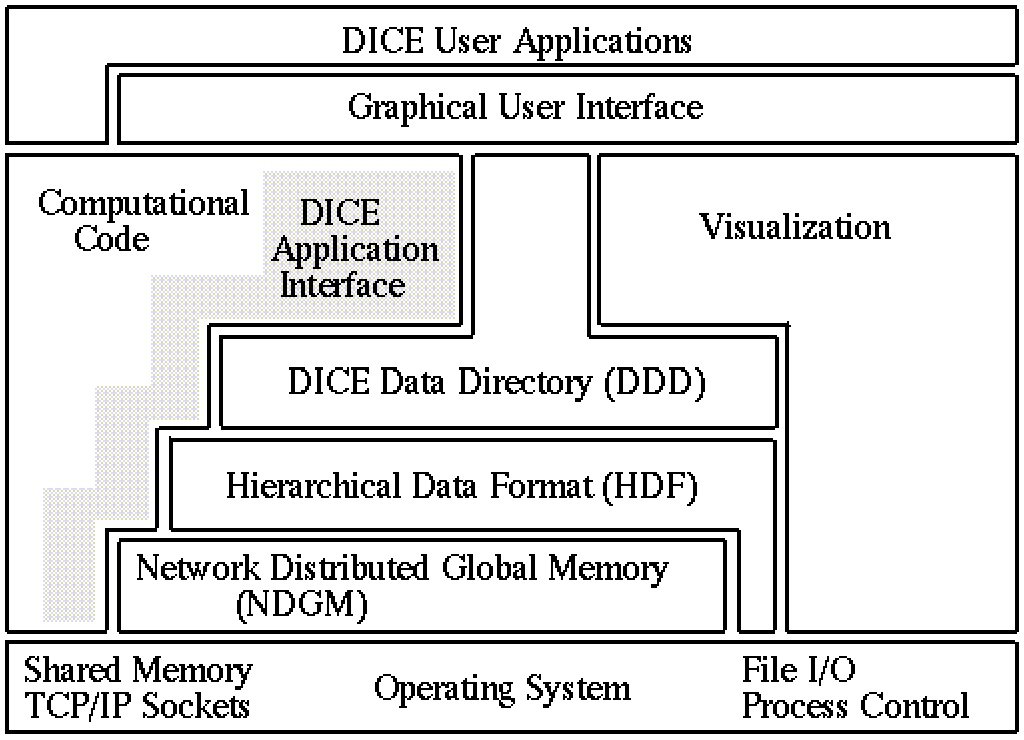

Figure 1. DICE Structural Overview.

the data buffer. These requests include information about what data variables need to be updated and about the frequency at which updates should take place. For example, CTH, a parallel structural mechanics code developed at Sandia National Laboratories, can produce data representing a wide variety of material attributes such as density, void volume and pressure for a large number of materials. Through a graphical user interface, these materials and attributes can be individually selected as data that should or should not be updated in the buffer.

Another manner in which the computation interacts with the environment is through input files. DICE applications may contain extensive graphical user interfaces for generating complex input files required by many HPC codes. For example, ZnsFlow, a fluid dynamics code being developed at the Army Research Laboratory, requires precise input parameters, especially for boundary conditions. Graphical user interfaces designed to handle this input greatly enhance the ability of a user to generate these files or change incorrect values in these files. The data analysis section of this paper will also describe how runtime visualization can be used to help find errors in these input parameters.

3. Storing and Accessing the Data from a Computational Process

As stated before, DICE purports the data to be the focus of an HPC application. The data organization component of this environment is composed of three layers. Each layer adds more structure and convenience utilities, yet maintains the flexibility needed to support data formats from almost any code.

At the heart of the data organization, is a unique heterogeneous shared memory system. Network Distributed Global Memory (NDGM) provides DICE with a physically distributed, logically shared, unstructured memory buffer (Clarke, 1997). NDGM is a client-server system that consists of multiple server processes and an Application Programmers Interface (API) for clients. Each server maintains a section of a virtual contiguous buffer, and handles requests for data transfer and program synchronization. Clients use the API to transfer data in and out of the virtual buffer, and to coordinate their activity.

NDGM provides a distributed, heterogeneous unstructured buffer. To provide some structure to this buffer, DICE uses the “Hierarchical Data Format” (HDF) from the National Center for Supercomputing Applications (NCSA). HDF, a well-known and widely used format, is designed to allow an orderly access to both structured and unstructured datasets. All access is accomplished through a well-defined application programmer’s interface. HDF is designed to access data via disk files. DICE alters some of the low level access routines to allow HDF to access NDGM as well as disk files. The only impact on the HDF API is the addition of the “domains”. HDF uses filenames in several of its API routines. If these filenames are prefixed by "NDGM:", the HDF data refers to NDGM. For completeness, prefixing "FILE:" to the filename refers to a disk file. If the domain is omitted, a disk file is assumed. This new HDF allows a single application to store data on disk, in distributed memory, or in both.

HDF defines a full-featured data format for structured and unstructured datasets as well as groups of data. However, it does not place restrictions on the organization of these datasets. To simplify access, DICE adds a convenience layer on top of HDF called the "DICE Data Directory" (DDD). Modeled after the UNIX filesystem, DDD provides facilities for mounting datasets and making sub-directories to organize complex data. DDD provides for structured datasets, unstructured datasets, and directories. In addition, DDD provides a “reference” dataset that points to a subsection of previously defined data. In this fashion, a single data file can reside on disk, in memory, or in both and contain many different types of data.

In addition, graphical tools have been developed to provide the user with a drag and drop facility for easily selecting and retrieving the data in DDD. Objects, known as "sources", usually represent a DDD item. Sources contain all information necessary to retrieve the underlying DDD item, but not the actual data. Objects are graphically displayed as icons, that can be selected, dragged and dropped onto targets in other window of the interface. Targets generally represent functions that operate on DDD items. For example, there is an information target that will display name, dimensions, and other pertinent information about any source that is dropped.

Through the use of NDGM, HDF, and DDD, the data organization of DICE provides a level of abstraction for enormous distributed datasets. Computational codes, visualization tools, and user interfaces can all access this data in a well-defined method without severely limiting performance. These utilities allow the data to be physically located to optimize the performance of specific components of the system, while maintaining consistent access methods through graphical interfaces.

4. Analyzing the Data with State-of-the-Art Visualization Techniques

DICE utilizes the Visualization Toolkit (vtk) for data analysis. Vtk is a set of tools and algorithms designed to facilitate the visualization process. It is freely available, well supported, widely used, and not specifically geared towards any one computational area. Intuitive graphical tools driven by complex scripts have been developed to provide easy access to the extensive functionality of vtk. Targets, as described in the previous section, are used as the mechanism for providing data input to the various visualization algorithms. These targets are combined with other graphical tools, such as sliders and buttons, to allow the user to specify any other necessary or optional information needed to generate a visual representation of the data. Underlying scripts obtain the actual data from the DDD and control the vtk rendering details, such as filtering and mapping. Thus, DICE transforms vtk into a ‘turn-key’ visualization package. By managing the vtk visualization network, DICE frees the end-user to concentrate on the actual analysis of the scientific data, and not the visualization details.

In addition to vtk, DICE also supports other visualization packages. A native DICE visualizer is available for unusual visualization needs. This visualizer provides direct access to how the graphics window is rendered, enabling huge datasets to be visualized via a composite rendering method. The visualizer uses OpenGL or MESA, depending upon the graphics abilities of the platform on which it is running. OpenGL will take advantage of available graphics hardware, while MESA is a graphics library that provides a totally software driven rendering system. MESA allows DICE applications to perform background renderings without connecting to any graphics sub-system, which can be particularly useful when graphic images are needed, but interactivity is not required. In addition to this native visualizer, DICE also includes data conversion utilities to some commercial visualization packages like EnSight, from Computational Engineering International (CEI), Inc. Similarly, data conversion utilities for some general chemistry packages will soon be added.

A powerful capability of DICE is the ability to perform runtime visualization. This feature exists because of the NDGM layer in the data organization component of the environment. Visualization routines can access the data in NDGM remotely, and in most cases, allow users to analyze runtime data from the comfort of their desktop. ZnsFlow, as with most typical fluid dynamics codes, requires precise input specifications. Small errors in these specifications can produce unusable output and consume valuable time on supercomputing resources. By visualizing data during the early stages of a fluid dynamics computation, these errors can often be detected, and the process can be killed before too much time is wasted. DICE uses scripts and graphical interfaces to pull together existing visualization software with its unique NDGM to make runtime visualization features available to virtually any code.

5. Using Scripts and Graphical User Interfaces to Seamlessly Connect the Components

A major portion of DICE is dedicated to providing the user with a flexible but powerful graphical user interface and a method for gluing together separate pieces of an entire application. DICE uses the Tcl language for a majority of its scripting needs. Tk and some popular Tcl/Tk extensions provide a well-rounded set of graphical widgets.

The Tcl scripting language is used to integrate the various tools of an entire application. This scripting language allows modular complex interfaces to be rapidly prototyped. However, some functions of an application require more efficient servicing than what a scripting language can provide. For these time critical functions, Tcl allows for functions written in a system programming language, such as C, to be loaded into the interpreter at runtime. Support for NDGM, HDF, DDD, and vtk have been included in this fashion. Thus, DICE maintains the speed for critical functions while benefiting from the reusability and rapid prototyping features of a scripting language.

Tk provides DICE with a base set of graphical widgets such as buttons and labels. Tix and BLT are two popular extensions to the Tcl/Tk combination that greatly expand this widget set by providing convenience widgets like fileboxes and hierarchical tree listings, a drag and drop facility and 2D graphing tools. These utilities allow complex graphical interfaces to be developed. For example, a 2D plotting widget to support line graphs within an interface allows the user to easily re-graph subsets of the data, or to query specific data points. This plotting widget is combined with a spreadsheet widget to give the user a complete data query interface.

All the significant components of DICE use scripts and graphical interfaces to interact and help coordinate their processes. Most of these scripts and graphical tools are not specific to any one particular application. Thus, they can be re-used to allow for the rapid development and integration of new DICE applications. Figure 2 shows an example interface for a typical DICE application.

6. Conclusions

DICE uses a unique data abstraction method to allow multiple application pieces to be combined into a single application, from the user's perspective. While single, monolithic applications that combine all features may perform better due to their tightly integrated nature, the programming effort necessary to make them attainable is quickly moving beyond the means of most organizations. By using a portable, powerful scripting language and reusable modules, complete applications can be designed. Each application incorporates a unique computational code with common data organization utilities and visualization tools. Through features such as runtime visualization, DICE facilitates the process of high performance computing, thus increasing the productivity of scientists and engineers. In addition, DICE also helps to reduce the need for expensive, redundant high performance computing access by taking advantage of existing hardware and network infrastructures.

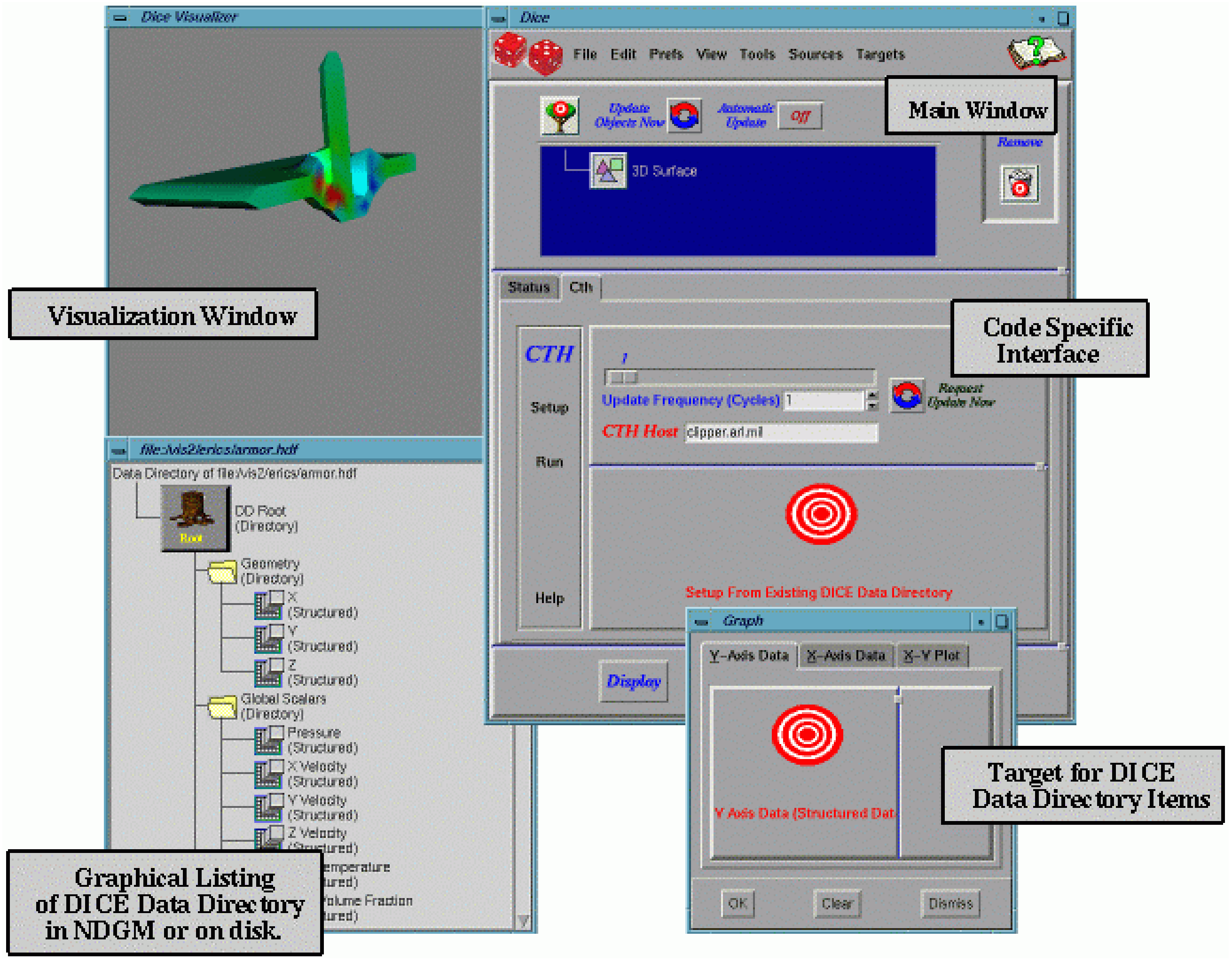

Figure 2. Example Interface for a DICE Application.

Acknowledgments

Portions of this work were performed under Contract DAHC94-96-C-0010 and Contract DAAH04-95-C-0008.

References

1. Clarke J.A., “Emulating Shared Memory to Simplify Distributed-Memory Programming," *IEEE Computational Science & Engineering*, Vol 4, No. 1, pp55-62, January-March 1997.

Bibliography

Clarke J.A., *Network Distributed Global Memory for Transparent Message Passing on Distributed Networks*, ARL-CR-173, Army High Performance Computing Research Center, US Army Research Laboratory, Aberdeen Proving Ground, MD, October 1994.

Laird, C. and Soraiz, K., "Choosing a Scripting Language," *SunWorld*, October 1997. Available on the World Wide Web at http://www.sunworld.online/swol-10-1997/swol-10-scripting.html.

Ousterhout, J.K., *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Co., Reading, MA, 1994.

Schroeder, W., Martin, K., and Lorenson, B., *The Visualization Toolkit: An Object Oriented Approach*, Prentice Hall PTR, Upper Saddle River, NJ, 1998.